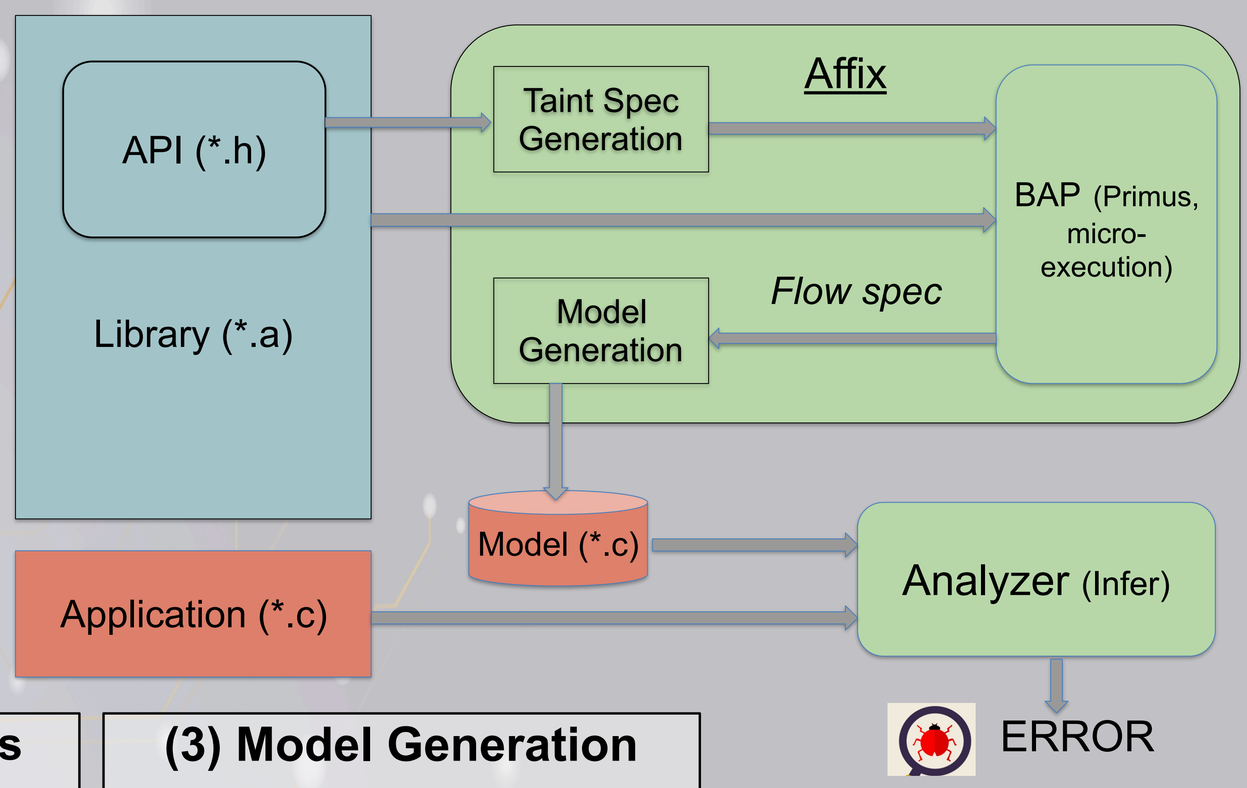
Affix: Binary Model Generation for Static Analysis

Correct Computation Michael Hicks mwh@correctcomputation.com

Static analysis tools, such as Facebook's Infer [1], require the source program. Developers often do not have access to the source code of the libraries on which their application depends, e.g., a shared library (`*.so`) distributed through a package manager. **Affix** is a tool for generating models (C source code) of binary libraries. Developers may use the models generated by Affix in their static analysis, thereby improving precision and accuracy.



(1) Taint Spec Generation

First, we generate BAP taint specifications that determine which values should be marked as sources and sinks of taint at runtime.

For example, to track the memory behavior of an API function (in *.h) we observe (a) which of its arguments may flow to `free` and (b) calls to `malloc` which are returned.

To achieve (a) we mark arguments as taint sources, and `free` as the taint sink. To achieve (b) we mark the return value of `malloc` as the taint source and function arguments / return value as taint sinks.

(2) BAP Taint Analysis

The Binary Analysis Platform (BAP) [2] has a micro execution framework called Primus. We leverage the dynamic taint analysis plugin for Primus to identify flows from taint sources to taint sinks (as specified in (1)). These flows are represented as source, sink pairs.

BAP will lift the library binary (`*.o` / `*.so`) into its own IR. The Primus micro execution framework then interprets this IR beginning execution at the entry point for each API call (i.e. there is no harness required). Furthermore, Primus will execute branching instructions nondeterministically, allowing analysis of multiple paths through the function.

(3) Model Generation

For each data-flow fact that Affix receives from (2), it will synthesize a C statement that emulates the data-flow.

For example, in the example below we receive the data-flow facts:

[malloc] return -> [make] return
[destroy] csort -> [free] ptr

and generate the model `lib-model.c` below.

The `add` function exhibits no memory behavior, so we synthesize an arbitrary integer return value.

Ongoing Work

We have successfully generated models for the bi-abduction and Quandary checkers for the Infer static analyzer.

We are currently scaling up our approach to support more features (nested pointers, structs, global variables).

We have tested our approach against modest libraries in open source C programs such as `sqlite` and `wget`.

lib.c

```
int *make(int lim) {
  return calloc(lim, sizeof(int));
}

int add(int *csort, int lim, int num) {
  int num_ok = 0 <= num && num < lim;
  if (!num_ok) { return 1; }
  csort[num] += 1;
  return 0;
}

void destroy(int *csort)
{ free(csort); }
```

gcc -c → lib.o

lib-model.c

```
int *make(int lim)
{ return malloc(0); }

int add(…)
{ return rand (); }

void destroy(int *csort)
{ free(csort); }
```

True Positive (`lib-model.c`)
The model generated by Affix makes the library's memory behavior evident, enabling Infer to find the memory leak in `app.c`.

⊢ ✗ `app.c:7: error: MEMORY_LEAK`

app.c

```
#include "lib.h"
int main(int argc, char *argv[]) {
  int *csort = *make(10);
  int r, err;
  for (int i = 0; i < 100; i++) {
    r = rand () % 10 + 1;
    err = add(csort, 10, r);
    if (err) { return err; };
  }
  destroy(csort);
  return 0;
}
```

⊢ ✗ `app.c:7: error: MEMORY_LEAK`

False Negative (`lib.o`)
Infer ignores the effects of calls into a binary library. It fails to find the memory leak in `app.c`.

⊢ ✓

References

[1] Infer static analyzer. Retrieved April 26, 2019, from https://fbinfer.com/
[2] Binary Analysis Platform. *GitHub*, 22 Apr. 2019, github.com/BinaryAnalysisPlatform/bap.

THE NINETEENTH ANNUAL HIGH CONFIDENCE SOFTWARE AND SYSTEMS CONFERENCE
Annapolis, MD | April 29 - May 1, 2019